第 41 卷 第 3 期 天 文 学 进 展 Vol. 41, No. 3
2023 年 9 月 PROGRESS IN ASTRONOMY Sept., 2023

doi: 10.3969/j.issn.1000-8349.2023.03.10

基于文件的大规模星表高效索引与融合

张琦乾[1,2]，樊东卫[1,2,3]，崔辰州[1,2,3]

(1. 中国科学院 国家天文台，北京 100101； 2. 中国科学院大学，北京 100049； 3. 国家天文科学数据中心，北京 100101)

摘要： 大规模星表的快速检索是交叉证认、多波段数据融合、暂现源搜寻等任务实现的基础，尤其是大视场暂现源搜寻需要在一个曝光周期内完成观测结果与大规模星表的检索与交叉证认，以发现正在变化的天体。现有的大规模星表通常包含数十亿天体，为了在有限内存的情况下对其进行快速检索，提出了一套解决方案。通过使用基于 HEALPix 的多分辨率动态划分算法，能够将星表按照不同天区天体密度切分成大小合适且均匀的星表文件；进而在开源序列化组件 Protocol Buffers 基础上设计出一套针对星表的序列化方案，作为星表切分和检索时的中间存储介质，以尽可能提高检索时的速度。还尝试应用 Peano-Hilbert 编号代替 HEALPix 原有 Z 型编号顺序遍历星表，提高了缓存命中率，实现了对大规模星表的高效融合，方便对数据的后续利用与研究。

关 键 词： 星表；HEALPix；交叉证认；虚拟天文台；天文软件

中图分类号： P129 **文献标识码：** A

1 引 言

随着天文观测设备的不断发展，人类对宇宙的探索能力也不断提高。特别是最近几十年来，计算机和电荷耦合器件 (charge-coupled device, CCD) 在天文学领域的广泛使用使得人类从过去的肉眼观测、胶片观测进入到数字化信息时代。多个大规模数字巡天项目预期能够获得的数据量远远超过人类手工处理的极限。例如建设在智利的 8.4 m 口径望远镜薇拉·鲁宾天文台 (Vera C. Rubin Observatory Legacy Survey of Space and Time, LSST)[1] 计划从 2022 年开始开展为期 10 a 的南半球 18 000 平方度巡天。在 10 a 中，每一个天区将会被巡

收稿日期： 2022-10-10； **修回日期：** 2022-11-30

资助项目： 国家自然科学基金 (12273077, 12103070)；国家重点研发计划 (2022YFF0711500)；国家自然科学基金委员会-中国科学院天文联合基金 (U1931132)；中国科学院网信专项 2022 年度应用示范项目 (CAS-WX2021SF-0204)。

通讯作者： 樊东卫，fandongwei@nao.cas.cn

访 1 000 次以上，获得 400 亿天体的观测数据以及一个空前规模的动态时域巡天数据库。泛星计划 (Panoramic Survey Telescope and Rapid Response System, Pan-STARRS)[2] 使用 4 个口径为 1.8 m 的望远镜组成阵列。

一方面，这些大规模巡天观测将产生数以太字节 (terabyte, TB) 乃至拍字节 (petabyte, PB) 量级的数据。例如，Pan-STARRS 基于第一台望远镜 Pan-STARRS1 (PS1)，该项目已发布 DR1、DR2 两批数据。在 2019 年 1 月 28 日发布的 PS1 DR2 星表中，ObjectThin 表包含 105 亿行数据 (约 5.4 TB)，StackObjectThin 表包含 34.7 亿行数据 (约 1.2 TB)。这些数据需要及时处理与分析。另一方面，其中的暂现源搜寻类巡天每次观测得到的数据又需要及时与之前大量的观测数据交叉证认，以判断是否有变化的天体。

前人在星表检索方面的很多工作都基于数据库而实现。对于天体坐标这类空间二维数据而言，若直接将天体信息存入数据库，在使用结构化查询语言 (structured query language, SQL) 进行锥形检索时，需要通过复杂的三角函数计算各个坐标之间的角距离[3]。少部分数据库如 PostgreSQL 有 PostGIS 等扩展组件支持对空间数据的检索，但是这些组件为地理信息系统 (geographic information system, GIS) 所设计，更多地用于处理地理上的三维点线面等复杂区域，而天文星表中的天体多为二维点源，且涉及天文坐标与地理坐标的转换等操作，在天文上少见应用。

更普遍的做法是使用伪二维球面索引算法将天球划分成多个子区域，然后将每个区域赋予一个 id 编号，星表中的天体以所在区域的 id 为主键存入数据库中。在检索时根据坐标计算区域 id 读取该区域的天体，再计算距离。目前球面索引算法有很多，例如条带算法 (Zones Algorithm)[4]、分层三角网格 (Hierarchical Triangular Mesh, HTM)[5]、四叉树立方体 (Quad Tree Cube, Q3C)[6]、多级等面积同纬度划分法 (Hierarchical Equal Area isoLatitude Pixelisation, HEALPix)[7] 等。Berriman 等人[8] 使用 2MASS 全天源星表 (two micron all sky survey all-sky point source catalog) (约 4.7 亿天体) 和未合并的 HSC 星表 (non-merged Hubble source catalog) (约 3.83 亿天体) 测试了 HTM、HEALPix 的不同层级分别在 Solaris、Windows、Red Hat Linux 平台下的 PostgreSQL 数据库和 SQL Server 数据库的检索速度。Szalay 等人[9] 开发的 SkyServer 采用微软的 SQL Server 数据库和 HTM 算法，实现了对斯隆数字巡天 (Sloan Digital Sky Survey, SDSS) 数据的在线检索和访问。这也是第一次使用商业数据库实现了大规模天文数据管理与访问[10]。

星表检索的一个重要应用就是实现对多个星表的交叉证认与融合，高丹[11] 通过使用 HTM 和数据库实现了对数十万量级星表的检索与交叉证认，但没能解决划分边界附近天体的漏源问题。Du 等人[12] 通过同时使用 HEALPix 和 HTM 以及数据库实现了对星表的交叉证认，缓解了漏源问题。Boch 等人[13] 通过使用 KD-Tree 和 HEALPix 实现了在线交叉证认平台 CDS-Xmatch，该平台允许用户在网页上传自定义星表与平台自带或之前上传的星表进行交叉证认。

前人的工作已经证实使用数据库可以实现较高效的星表检索。从图 1 的对比中可以看出，在使用伪二维球面索引算法对星表进行基于坐标的检索时，数据库的作用是存储天体信息和对应 id 编号，在检索时需要找到所有与待检索 id 相同的天体，交由上层软件计算角距

离并加以过滤后，才能得到最终符合要求的结果。根据层级选择的不同，可能会有成千上万个天体都具有相同的区域 id，在这种特定场合下，若不使用数据库，而是直接将相同 id 的所有天体保存在同一文件中，检索时只需读取对应的文件即可得到该 id 编号下所有的天体，也可以实现对星表的索引与检索功能。

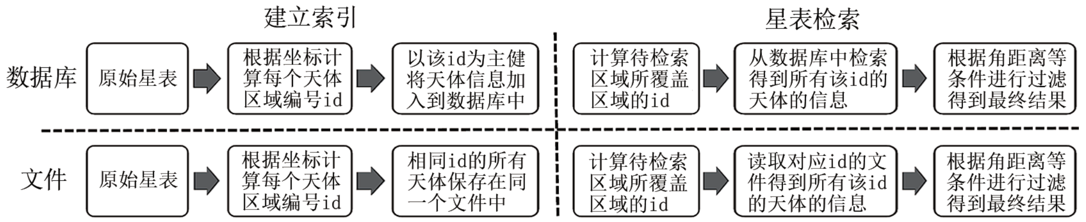


图 1　使用数据库与文件对星表进行索引与检索的流程对比

在划分时区域编号对于数据库仅是一个用于建立索引的数字，但是 HEALPix 等类似算法的不同层级、同一层级的不同区域之间的 id 编号往往有一定的关联，在使用文件作为星表的存储介质时，可以利用这些特点做一些针对星表的优化，提升索引与检索的效率。此外，与数据库这类商业软件不同，文件具有无需授权，通用性更好，也无需提前安装配置，易于维护等优点。在诸如偏远台站等较难维护，或者在单片机、小型探测器等这类性能较低，难以运行数据库的场合，文件都具有独特的优势。因此，本文尝试使用文件作为星表检索时的存储介质，并加以优化，以提高在星表交叉证认或融合时的检索效率。

本文选择了 HEALPix 作为划分的基础，该方法由 Górski 等人提出，能够根据预先设定的层级将天球切分成多个面积相同的子区域，称之为“像素”，每个像素都有对应的编号。层级越大，像素越多，单位像素面积越小，对天球划分的分辨率越高。HEALPix 最开始应用于宇宙微波背景辐射的研究与分析，如今已经在天文学研究中广泛使用。Pineau 等人[14]使用 HEALPix 作为划分方案对星表进行了交叉证认测试。原始 HEALPix 只能以固定分辨率划分球面，之后有各种多分辨率的 HEALPix 解决方案被提出，例如多层级覆盖天区 (multi-order coverage map, MOC)[15, 16]、多分辨率 HEALPix (multi-resolution healpix, MRH)[17]、mhealpy[18]等。许允飞[3]通过使用 MOC 建立的多层级覆盖天区四叉树 (multi-order coverage tree, MOC-Tree)，从而对不规则区域星表进行索引。Martinez-Castellanos 等人[18]发布的 mhealpy 是一套多分辨率 HEALPix 及可视化的 Python 包，支持两种多分辨率模式，其中一种允许用户通过自定义函数选择当前区域的划分层级，但是受到可视化的限制，某个区域只能被一个层级所覆盖。而星表的切分不需要考虑可视化相关的限制，本文尝试允许多个层级的像素覆盖同一区域，实现对星表的均匀切分。

天文星表在计算机中存储和交换的格式以逗号分隔值 (comma-separated values, CSV) 和灵活图像传输系统 (flexible image transport system, FITS)[19]为主，此外常用的还有 VOTable 和一些机构制定的文本或二进制格式。CSV 格式的星表以纯文本格式存储，使用逗号或其他符号进行分隔，使用文本编辑器即可读取和修改内容。程序使用 CSV 存取星表的天体坐标时需要进行字符串和浮点数的转换，对存取大规模星表的速度有一定的影响。

FITS 是由国际天文学联合会定义的信息交换格式，最初是为了传输图像而设计[20]，之后扩展到星表等更复杂的数据格式；FITS 格式的星表以二进制格式按列存储，读写需要依靠第三方软件或库。

在检索星表时仅需要读取坐标等少量数据，也不需要考虑可读性。另外这些文件仅供检索程序运行时使用，不需要对外分享。可以考虑通过定制专属的序列化协议，对这些数据序列化，以减少中间格式转换的次数，达到更好的存取性能。

巡天观测扫描的天空区域通常是连续的，前一次检索的部分数据可为下一次检索所用。通过选择适当的缓存策略，可以减少文件读取次数，提高数据的复用率，并实现更快的连续检索。除此之外，在执行多个大规模星表融合等需要对大量天体交叉证认的任务时，通过选取合适的遍历顺序，可以提高缓存的命中率，提高交叉证认的速度。

针对在基于文件的情况下如何对星表高效检索的问题，本文提出了一套解决方案。本文第 2 章研究如何在有限内存下将大规模星表切分成合适且均匀的星表子集；第 3 章研究如何对切分后的星表子集做高效检索；第 4 章探讨如何使用开源序列化库对星表高效存取；第 5 章将讨论实现多个大规模星表的交叉证认与融合的方法；第 6 章将对前几章提出的方法做实验测试；第 7 章进行总结。

2 星表索引及星表文件的切分策略

从一个包含有 X 个天体的星表中搜寻 Y 个特定天体，若不对星表做任何处理，直接从头到尾按顺序查找，最坏情况下需 $X \times Y$ 次检索。一个改进方法是用诸如 KD-Tree 等多维数据结构建立索引加快检索时的速度。受制于当前计算机体系结构，这些数据结构建立和检索只能在内存中进行。但对于动辄数 TB 的星表，一方面计算机内存难以将星表全部读入到内存，另一方面从硬盘中将星表完整读取一遍也会消耗大量时间。为了加快查找速度，需要针对性的优化。

对于星表来说，有以下两种基本的基于坐标的查询操作：

1) 最近邻天体检索，即给定坐标，检索星表中与之最为接近的天体；

2) 锥形检索，即给定坐标和距离阈值，检索星表中所有距离该坐标小于该阈值的天体子集。

这两种操作只需要检索给定坐标附近区域内天体，并不需要将整个星表都读取并检索一遍。

因此，在检索前可以将星表按照天体的坐标执行切分，把邻近的天体切分到同一个文件中，并给切分到的每个文件赋予一个特定的编号。在检索时只需要根据待检索坐标等信息，计算所覆盖到区域的编号，并根据该编号读取并检索相应的文件，这样就可以避免读取整个星表，从而大大提高查询效率。

切分出的星表子集应做到大小合适且尽量均匀。若单个文件内天体数量太多，则检索时读取时间和建立索引的时间较长，影响检索的效率；若数量太少，则每次检索需要加载大量

的星表文件，而且硬盘对小文件的随机读写速度远远小于顺序读写，检索时需要跨越多个星表文件，极限情况下甚至会退化成顺序检索，造成效率的下降。

因而，需要对切分后单个星表文件天体数量上下限做出限制，以最大化星表检索效率。

2.1 多分辨率 HEALPix 星表切分算法

在星表中，天体的位置坐标作为必不可少的信息之一，适合作为切分的首选依据。但直接按照赤经和赤纬的数值线性划分得到的区域面积不等，难以实现均匀的划分，本文选择能将天球等面积划分的 HEALPix 作为切分的基础。

虽然 HEALPix 对天球的划分是等面积的，但是天体在天球中的分布往往是不均匀的。例如，对于一个以河内恒星为主的星表来说，银心方向的天体密度往往大于反银心方向或者垂直于银道面方向。原始的 HEALPix 只能以一个固定的层级划分天球，导致不同区域天体数量相差较大，以此切分得到的星表文件也会有较大的体积差异，影响对星表处理的效率。为了保证切分出来的星表尽可能均匀，本文提出了多个 HEALPix 层级共存的星表切分算法，能够根据区域天体的密度动态选取一个合适的切分层次。

HEALPix 首先将天球划分成 12 个等面积的大区域，并在此基础上根据一个设定的层级 (设为 k) 将整个天球划分成 12×4^k 个等面积的小区域。k 值越大，划分精度越细，分辨率越高。$k = \{0, 1, 2\}$ 时的划分如图 2 所示。

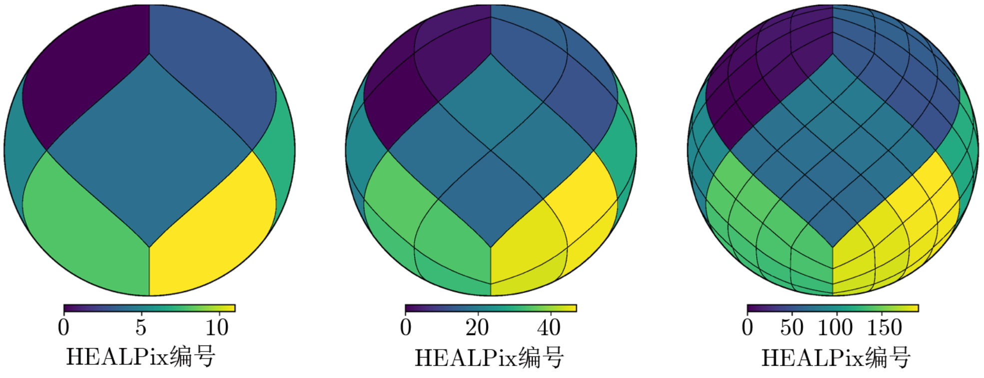


图 2　从左至右分别为 HEALPix 在 $k = 0$, $k = 1$, $k = 2$ 时对球面的划分情况

HEALPix 支持 RING 和 NESTED 两种编号模式。RING 模式下，从北极到南极，相同纬度的区域形成一个环，在环内按照顺序依次编号。NESTED 模式下，将第 i 层编号为 $n_{\mathrm{pix},i}$ 的区域继续划分成 4 份，即可得到 4 个第 $i+1$ 层的区域，编号范围是 $[4 \times n_{\mathrm{pix},i}, 4 \times n_{\mathrm{pix},i} + 3]$。这样的特性非常适合应用四叉树。因此，本文选择在 NESTED 模式下使用四叉树作为联系起不同层次区域之间的索引方案。

四叉树是一种基于空间递归分解思想的分层数据结构。树的根节点可以代表总的区域，将该区域划分成 4 个子区域后，每个子区域都可以用四叉树的一个子节点来表示。每个子区域还可以再细分为四个更小的区域，这个过程可以一直递归进行，直到达到所需的分辨率[21]。四叉树与 HEALPix 层级之间的映射关系如图 3 所示。

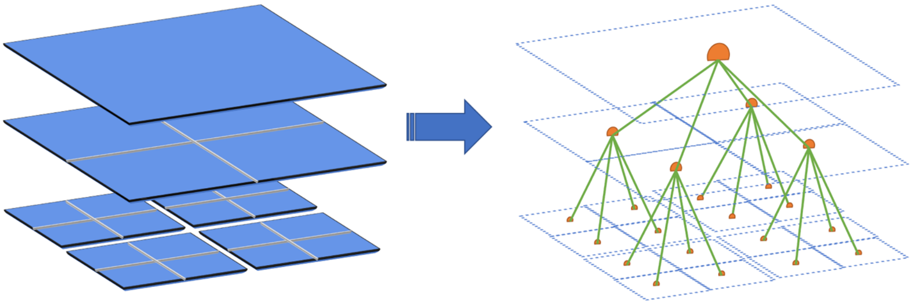


图 3　不同层级的相同区域可以用四叉树的不同节点来表示

由于 HEALPix 不同层级区域都是由最初的 12 个区域划分而来，所以可建立 12 棵四叉树，每棵树的根节点代表 $k=0$ 时的 12 个区域其中之一。树第 i 层的节点，与 $k=i$ 时的划分区域一一对应。树的每个非叶节点都有 4 个子节点，指向下一层的 4 个子天区。每个子节点的 $n_{\rm pix}$ 编号与父节点有相同的二进制前缀。这样就实现了四叉树与 HEALPix 多个层级划分区域的一一对应，可以使用对树的相关操作来实现对天区的有序访问。

由于 HEALPix 在不同层级下对区域均是从 0 开始编号，因此不同层级之间的区域编号会产生冲突，这种二维编号方式会影响检索的效率。本文采用 MOC 中的 NUNIQ 编码方式，赋予不同层级的区域一个唯一的编号。层次为 k，编号为 $n_{\rm pix}$ 的区域的 UNIQ 编号可通过下式计算：

$$N_{\rm UNIQ}=4\times 4^{k}+n_{\rm pix}\quad (0\leqslant n_{\rm pix}\leqslant 3\times 4^{k+1})\quad , \tag{1}$$

逆运算为：

$$k=\lfloor \log_2(N_{\rm UNIQ}/4)/2\rfloor\quad , \tag{2}$$

$$n_{\rm pix}=N_{\rm UNIQ}-4^{k+1}\quad . \tag{3}$$

由于大规模星表大小 N 往往会远远大于可用内存容量 M，而程序运行时所能读取的星表又不能大于 M。如图 4 所示，此时就需要将星表事先分割成 $\lceil N/M\rceil$ 块局部星表，每块大小都小于 M。这样，程序每次都可以将一块局部星表完整读入到内存执行切分，之后清空内存并读取下一块局部星表。而对多个局部星表的切分后直接的合并，并不等同于对原始星表的切分结果，需要再次利用四叉树归并。所以本文的星表切分算法分成两步：第一步是对星表的局部切分，第二步是将多个局部切分的星表归并。

2.2　星表的局部切分

首先需要指定一个参数 $k_{\max}$，代表所能使用的 HEALPix 的最大层级，并作为四叉树的最大层级。但 $k_{\max}$ 越大，程序运行时消耗的内存也越多。

第一步是四叉树的建立与星表的读取。如上文所述，建立 12 棵四叉树，每棵树有 $k_{\max}$ 层，第 i ($i\in[0,\ k_{\max}-1]$) 层有 4^i 个节点。12 棵树共计 $4^{k_{\max}+1}-4$ 个节点，每个节点都有一个 UNIQ 编号并与天球上一个区域相对应。

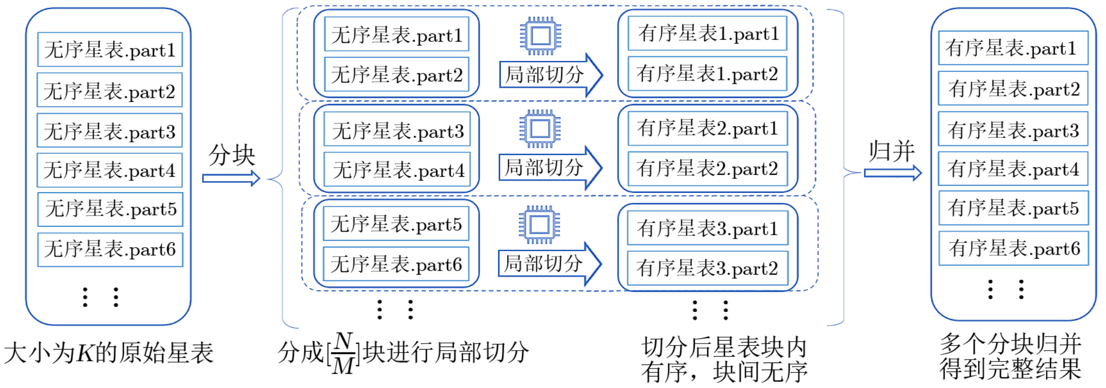


图 4　对远大于内存容量的无序原始星表的切分处理的过程

四叉树的每个节点内有一个变量 C 代表该节点对应区域内的天体数量，初始时为 0。每读取星表中一个天体信息，就根据它的坐标值，分别计算出该坐标在每个层级下的 HEALPix 的区域编号。并根据此编号计算对应的 UNIQ 编号，同时将对应的节点 C 值加 1。即实现了天体在四叉树中的索引。

当程序将一块局部星表读入到内存并用四叉树索引后，将执行切分操作。切分算法需要事先指定一个切分阈值 T，在不同切分模式下作为单个星表文件的天体数量上限或者下限，当遍历到某一节点时，根据该节点所覆盖区域内的天体总数与阈值 T 相比较，以确定下一步的操作。在递归切分时，会遇到以下 3 种情况：

1) 父节点的 C 值和 4 个子节点中的 C 值均大于 T，则继续递归遍历 4 个子节点；

2) 父节点的 C 值大于 T，且存在子节点的 C 值小于 T 的节点，处理方式见下文；

3) 已到达 $k_{\max}$ 层，则将当前节点内的天体切分，并停止向下遍历。

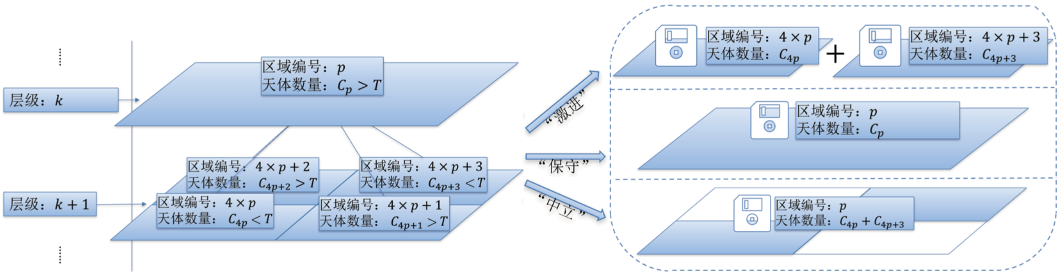
在星表分布不均匀的情况下，例如北半球地面望远镜受地理纬度的限制难以观测到南天区的天体，会出现南天区中根节点 C 值也小于 T 的情况，此时为特例情况，将根节点内所有天体切分即可。

对于情况 (2)，又可以分为以下 3 种处理方式，如图 5 举例所示。

1) “激进”方式。对 C 值大于 T 的子节点继续遍历其子节点；对小于 T 的子节点，则将子节点对应区域的天体切分至对应文件中，并停止该子节点的向下递归，即在图 5 例中将 $4\times p$ 和 $4\times p+3$ 的节点各自切分到两个文件中。该方式是 mhealpy 中所提供的 2 个切分方案之一，在本文中按照其特点将其命名为“激进”方式。

2) “保守”方式。只要 4 个子节点中存在 1 个 C 值小于 T 的节点，就将父节点对应区域的天体切分到对应文件中，并停止遍历其子节点，即将图 5 中编号为 p 的父节点切分。

3) “中立”方式。分别访问该节点的 4 个子节点，若该节点 C 值大于 T，则继续遍历其子节点，并得到一个返回值 r，代表了对应区域有 r 个天体被切分。若该节点 C 值小于 T，则不再遍历其子节点，而是将 C 的值作为 r 返回至父节点。父节点收到各个子节点的返回值后计算剩余未被切分的天体数目，若总数大于 T，则以父节点的名义将这些天体切分



注：使用 HEALPix NESTED 编号，假设 $k+1$ 层编号为 $4\times p$，$4\times p+3$ 的两个子区域内的天体数量少于阈值，其余区域大于阈值。右侧为“激进”、“保守”、“中立”三种切分方式得到的星表子集的编号和天体数量。

图 5　星表切分情况 (2) 时的 3 种处理方式

到对应文件中，并返回总数；若小于 T，则将已切分的数量返回至更上一层节点。依次类推，直到该节点总数大于 T，或到达根节点时将天体切分到文件中。图 中假设 $4\times p+1$ 和 $4\times p+2$ 区域已全被切分。

不同处理方式之间的优缺点比较见表 1。经过对比不难看出，本文所提出的“中立”切分方式在理论上可以切分出最均衡的星表文件，适合作为实际应用时的首选切分方式。

表 1　不同切分方式的优缺点对比

类型	优点	缺点
保守	切分后单个星表文件的数目皆大于 T (特例情况时除外)。	当 4 个子节点中有一个小于 T 时，则将整个父节点一并切分输出，极端情况下切分出的文件可能会很大。
中立	通过选择仅切分大于阈值的区域，并通过增加返回值的方式将被切分的目标数量传递给父节点汇总。这种方式使得切分后星表内天体的数目会在 $[T, 4\times T]$ 之间 (特例情况时除外)，大小较均衡	某个天区可能同时被多个层级所覆盖，但该天区的目标只存在于层次最大的文件中。即虽然重叠，但不会重复。这一特点对星表的检索无影响，但在可视化时无法将所有区域绘制在一副天球图像上。
激进	切分后单个星表文件的天体数量严格小于 T，此时 T 的大小即为切分出的星表文件的数目上限。	可能会切分出大量小文件，相同 T 下切出的文件总量也是三种方式最多的。

在切分完成之后，清空内存中的星表，若星表未读取完毕，则继续读取星表，并重复上述切分步骤，直至所有星表读取完毕，完成星表的局部切分。

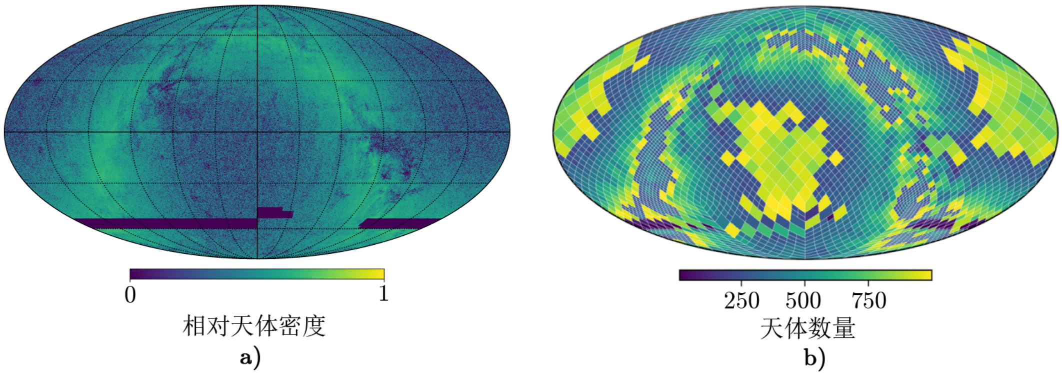
2.3　局部切分星表的归并

一个无序的星表经过 $\lceil N/M\rceil$ 次局部切分后，得到了 $\lceil N/M\rceil$ 块局部切分的星表文件集合，每块集合内有不定数量的星表文件，如图 4 所示。虽然块内的天体都按照所在区域划分到各自的文件中，但分块与分块之间却是互相独立的。因此需要归并各个分块的星表，得到对原始星表完整的切分结果。

同样是建立 12 棵对应的四叉树。从每棵树的根节点开始，按照深度优先的顺序遍历树的各个子节点。若某个局部切分的星表中存在该节点对应的星表文件，则读取到内存，并更新对应四叉树节点中的 C 值。之后依次递归遍历该节点的 4 个子节点，并接受各个子节点的返回值 (仅“中立”模式有返回值)。

当遍历完 4 个子节点并返回到当前节点时，则意味着以该节点为根的所有子节点均已被访问过，星表中该节点所在区域的所有天体均已被访问过，或留在内存中，或已经在遍历某个子节点时被切分到最终星表文件中。此时，与局部切分过程类似，根据切分模式、阈值 T 和当前节点所剩余的天体数量，判断是否执行切分。“中立”模式还需向父节点返回对应的值。此时切分得到的星表文件即为归并之后的星表文件。此过程不断进行，最后实现对局部星表的归并。

图 6 显示了未切分之前的星表和“激进”模式下动态切分后的星表。可以看到，天体密集的区域切分的精细程度比较高，天体稀疏的区域切分精细度较低，保持了每个切分区域天体数量的均衡。



注：以“激进”方式为例，阈值设置为 1 000，a) 图是原始星表的天体相对密度图，b) 图是切分之后，颜色代表该区域内的天体数量分布 (星表局部区域有数据缺失)。

图 6　星表动态切分示意图

3　切分后星表的检索

在将星表按照上述切分算法切分之后，每个切分后的星表文件都有一个 UNIQ 编号，代表了其所覆盖天区。在检索时，根据不同检索方式，将待检索区域转换为对应的 UNIQ 编号列表并读取对应的文件。

3.1　区域检索

由于“中立”模式下多重覆盖的特性，对于最高层级的某个区域来说，该区域内的天体只会存在于该区域所在的节点或距离该节点最近的有星表子集存在的上层节点中。而一个非最高层级的区域内天体可能会分散在多个层级的不同星表子集文件中，此时需要遍历该

区域的父节点和子节点查询。

如果想要在切分后星表中检索到某个区域，例如第 k 层，HEALPix 编号为 pix 的节点所覆盖区域内全部天体。首先从该 pix 节点开始，向上递归访问其父节点，直到访问到第一个存在星表子集文件的节点，读取文件内所有满足编号在目标区域内的天体到内存中。若 k 小于 $k_{\max}$，则还需要依次遍历该 pix 节点的所有子节点，并将子节点中存在对应的星表子集的文件全部读取，最终得到符合要求的天体信息。

若待检索区域是一个多边形，则可以将该多边形区域导入到 MOC 中，得到对应区域的 UNIQ 列表。再对该 UNIQ 列表中的每个区域进行上述的区域查询，并汇总，即可得到最终的检索结果。

3.2 对坐标的检索

若给定一个坐标和阈值，要求检索以坐标为圆心，阈值为半径内的所有目标，首先使用 HEALPix 自带的 query_disc 函数计算得到在最大层次下被该区域覆盖得到的 pix 列表，将列表中的每一个 pix 执行上述区域查询，并对已经读取的星表做好标记避免重复读取。

对于两个在天球上的坐标 (α_1, δ_1) 和 (α_2, δ_2)，他们之间的角距离 d 可由 Haversine 公式[22]给出：

$$d = 2\mathrm{arcsin}\sqrt{\sin^2(\frac{\delta_1 - \delta_2}{2}) + \cos\delta_1\cos\delta_2\sin^2(\frac{\alpha_1 - \alpha_2}{2})} \quad . \tag{4}$$

对内存中的天体分别计算并将与坐标的角距离小于阈值的目标加入到返回列表中，得到检索的结果。

而在最近邻检索时，待检索坐标与最近邻天体可能不在 $k_{\max}$ 层级下同一个区域中，所以需要读取与该区域相邻接区域内的所有天体。读取后通过 Haversine 公式计算与待检索坐标的距离，找到与待检索目标距离最近的天体。

4 数据的高效存取

数据在内存中通常是以对象、结构体、数组、列表等形式连续或离散地存储于不同的区域供 CPU 访问和操作。当内存中的数据需要被写入到文件或通过网络传输时，则需要对其进行编码 (也称之为序列化 (serialization)、编组 (marshalling)) 为字节序列，反过程则称之为解码 (或反序列化 (deserialization)、反编组 (unmarshalling))[23]。例如将内存中的天体以 CSV 格式保存到硬盘即是对星表序列化的过程，对 CSV 星表的读取则是反序列化过程。

在切分过程中，星表的每个天体都会被读取两次，写入两次，数十亿天体将会产生大量硬盘 I/O 并消耗大量的时间。一方面，由于星表的坐标等数据都是以浮点数的形式存储在内存中，如果简单地以 CSV 等文本格式存储，需要解析每行的各个表项以及对浮点数与字符串之间相互转换，由此产生的计算量和耗时也不容小觑。另一方面，在切分时，程序只读取天体的坐标，其他的物理参数则不必考虑。

为了尽可能地提升性能，本文以开源的高性能序列化组件协议缓冲区 (Protocol buffers,

Protobuf)[24] 为基础，并针对星表这一格式的数据定义了对应的序列化格式，实现对星表数据的高效存取。

Protobuf 能够将内存中数据按照事先约定的格式写入程序与程序之间，或者通过网络传递，也可以写入到磁盘保存。功能上类似于可扩展标记语言 (extensible markup language, XML) 和 JavaScript 对象简谱 (JavaScript Object Notation, JSON)。但是与之不同的是，Protobuf 需要事先约定好一套协议，称之为接口描述语言 (interface description language, IDL)，读写时需要持有相同的 IDL。数据按照 IDL 规定的格式以二进制的形式存储，并使用多种压缩算法，避免了数字与字符串之间的相互转换。并且支持 C++、JAVA、Python、Go 等主流的语言。

本文定义了星表在 Protobuf 中的传输格式，包括一个“Catalog”项用于储存星表信息，内容如表 2 所示，其中包括星表文件的文件名、UNIQ 编号、星表头信息、总行数信息等；同时内部有任意数量的“CatalogLine”项，用于存储天体信息，内容如表 3 所示，其中包括了天体的坐标、区域编号，分别以浮点数和整数进行存取。而天体的其他信息则保存为字符串类型，以减少对数据的解析时间。

表 2　使用 Protocol buffers **定义的** Catalog **项**

字段名	类型	说明
name	string	星表的名字
uniq	int32	星表内天体所在区域的 UNIQ 编号
total_lines	int32	星表的天体数量
header	string	星表的表头信息
line	repeated CatalogLine	星表中的天体，repeated 表示数量可为 1 至多个

表 3　使用 Protocol buffers **定义的** CatalogLine **项**

字段名	类型	说明
pix	int32	该天体所在区域的编号
ra	double	该天体的赤经
dec	double	该天体的赤纬
line	string	该天体的其他信息

根据定义的传输格式，Protobuf 可以自动生成不同编程语言下的序列化和反序列化的代码。这使得在星表切分、归并和检索时，能够根据需要，将内存中的星表序列化为 Protobuf 格式并保存到硬盘对应的文件中，以及从硬盘中读取文件并反序列化回内存，实现了对星表数据的高效存取。

5　多个大规模星表的融合

在暂现源搜寻时往往需要同时对多个不同波段的星表，或者多个不同观测时间星表进

行检索。为了加快检索速度，可以提前将多个星表合并，即对一个星表中的天体与其他星表交叉证认，找到对应天体并对各项参数加以融合，从而实现一次检索就可以获得多个星表中的数据。

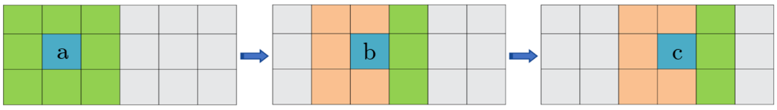
由于不同望远镜观测时的精度差异等多种原因导致相同的源在不同的星表中的位置不是完全相同的。一般来说，对于误差半径分别为 r_1、r_2 的两个星表，当两个天体的角距离 $d < 3\sqrt{r_1^2 + r_2^2}$ 的时候，就可以认为是同一天体[3]。位于 HEALPix 划分区域边缘的天体误差半径可能会覆盖到相邻区域，所以在计算某一个区域的距离时还需要读取与该区域相邻的几个区域内的天体，此时可以通过选择合适的遍历顺序和增加缓存以提升效率。

5.1 星表在内存中的索引

将天体按切分到不同的星表文件时，天体在文件内的顺序即切分时的读入顺序。若需要检索某个文件内的天体时，顺序检索效率不高，可以将该文件全部读入到内存，并在内存中建立 KD-Tree，实现二维检索，提高检索速度。

KD-Tree 是一种对空间多维数据高效索引的数据结构。KD-Tree 会在建立时在每层选择不同维度的数据计算中位数并作为索引节点，最终构建出一颗比较均衡的二叉树。而 KD-Tree 在检索时，则按照层数分别将不同维度的数据与树中的数据进行对比，从而能够高效检索高维数据。

根据之前的切分情况，以星表文件作为建立 KD-Tree 的单位，若某次检索的区域覆盖了多个星表文件，则需要在内存中建立多个 KD-Tree。此时可以将建立好的 KD-Tree 缓存到内存中，供之后的检索使用，避免重复读取，提高效率，如图 7 所示。而缓存大小不可能无限大，特别是在同时运行多个任务时，内存中的 KD-Tree 数量达到上限，则可以选择合适的缓存替换算法，例如先进先出 (FIFO)，或者最近最久未使用 (LRU) 等，替换缓存中的 KD-Tree。各种替换算法在计算机等领域已经被深入研究过，本文将通过实验探究不同缓存算法在星表证认时的性能差异以及最优缓存的大小。



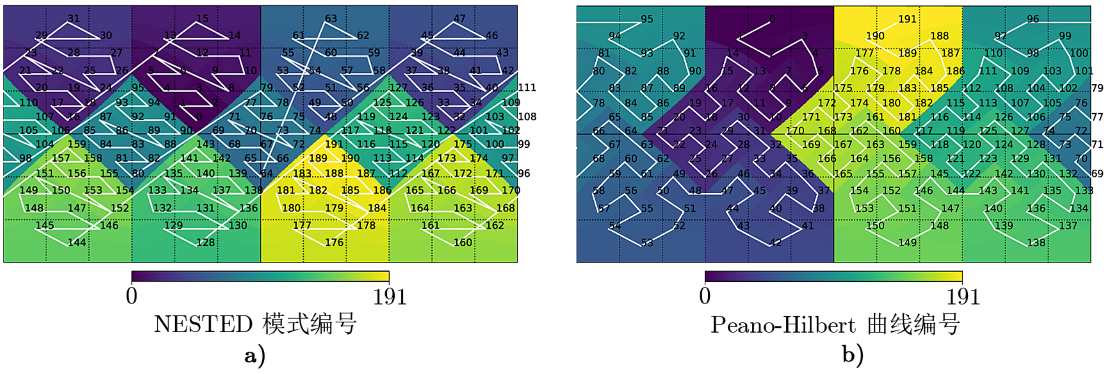
注：当对 a 区域交叉证认时，为了避免边界漏源问题，需要读取另一星表的 a 及周围 8 个区域的天体，并按照区域建立对应的 KD-Tree。可以将这些 KD-Tree 分别缓存在内存中，当对相邻的 b 区域证进行认时，仅需读取右侧 3 个区域并建立 KD-Tree 即可，对 c 区域证认时也是同理。

图 7 KD-Tree 检索示意图

5.2 使用 Peano-Hilbert 曲线遍历星表

在对 HEALPix 按照 NESTED 模式编号顺序遍历时，路径在空间表现为Z曲线的形式[25]。该顺序可以高效地为平面或空间中的一组点集构建四叉树或者八叉树[26, 27]。而 Z 曲线是在不同区域的边界处有比较大的转折，即会跳转到一个不相邻的区域中。在对星表遍历时往往会导致之前的缓存大量失效，降低效率。为了克服这个缺点，本文尝试在对星表遍历时使用 Peano-Hilbert 曲线[28]代替原有的 Z 曲线。该曲线通过对 HEALPix 的区域重新进行

编号，实现了相邻编号的两个区域在天球上也相邻，因此对星表进行遍历时会一直访问相邻的区域，能够充分利用已有的缓存。按编号顺序分别对 NESTED 自带编号和 Peano-Hilbert 曲线编号对星表进行遍历，得到曲线对比如图 8 所示。



注：a) 按 NESTED 模式编号遍历星表得到的 Z 曲线；b) 按 Peano-Hilbert 曲线编号遍历星表得到的曲线。

图 8　按 NESTED 模式编号顺序和 Peano-Hilbert 曲线编号顺序遍历星表示意图

6　实验分析

本实验使用 LAMOST DR7 星表作为测试数据，运行平台使用国家天文科学数据中心提供的云虚拟机，配置如下：

1) CPU 为 Intel(R) Xeon(R) CPU E5-2690 4 Core @ 2294.686 MHz；

2) 内存为 8 GB；

3) 硬盘 I/O 速度为 82.5 MB/s；

4) 操作系统为 Ubuntu 20.04.4 LTS；

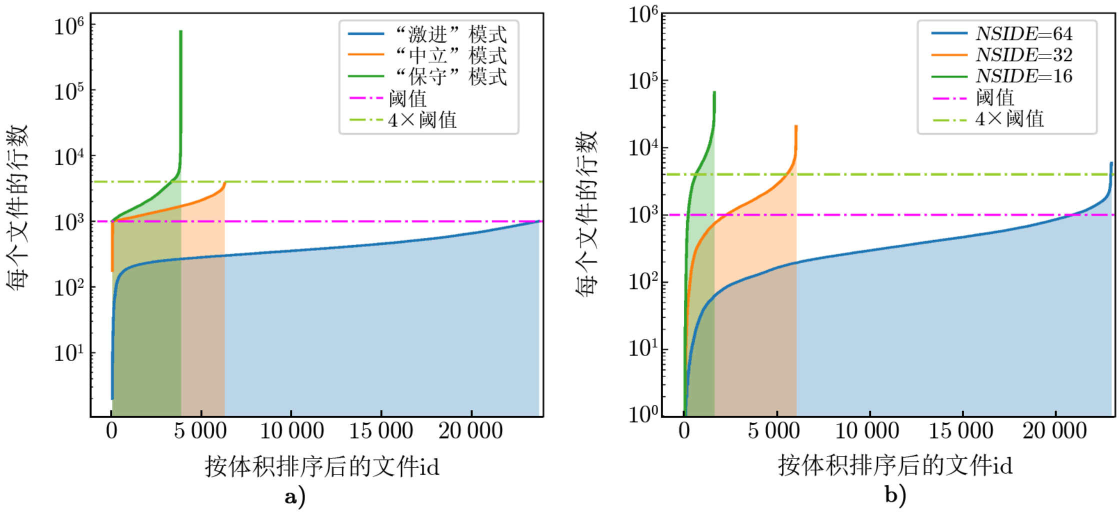
5) 编程语言为 C++、Python。

6.1　不同切分方式产生的星表文件对比

分别对测试星表进行“激进”、“中立”、“保守”模式的动态切分和按固定层级的切分。测试动态切分使用的切分阈值为 1 000，数据如图 9 和表 4 所示。

可以看出，“激进”方式切分出的星表的天体数量均小于阈值，大部分集中在 [200, 1 000] 之间。“中立”方式切分出的星表的天体数量绝大部分集中在 [1 000, 4 000] 之间；由于某些南天区的目标过少，导致极少量的文件小于阈值；同时，“中立”模式下切分出的星表的变异系数也是所有模式中最低的，说明星表的离散程度小于其他模式。“保守”方式切分出的星表均大于 1 000，特别是一部分星表的天体数量过大，最大约 10^6，其变异系数也是各个方式中最大的。三个固定层级切分的结果变异系数相差不大，皆大于“中立”和“激进”模式，小于“保守”模式。

切分出的结果与预期相符，“中立”方式切分出的星表较其他两种切分方式和固定切分



注：a) 是相同阈值下动态切分的三种方式做切分；b) 是使用相近的三个层级对星表做静态切分。

图 9　将相同星表按照不同方式切分并排序后的大小曲线

表 4　不同切分方式的数据对比

类型	星表文件总数	平均行数	行数最大值	行数最小值	标准差	变异系数
动态层级 (保守)	3 798	2 746.49	778 747	1 002	12 965.52	4.720 7
动态层级 (中立)	6 233	1 673.54	3 818	178	565.57	0.337 9
动态层级 (激进)	23 649	441.08	1 000	2	199.05	0.451 2
固定层级 (k=4)	1 578	6 610.39	65 792	1	6 016.63	0.910 1
固定层级 (k=5)	5 972	1 746.68	20 835	1	1 608.43	0.920 8
固定层级 (k=6)	22 891	455.69	5 917	1	421.35	0.924 6

更为均匀。

6.2　Protobuf格式与格式星表的读取速度对比

本文将 LAMOST DR7 星表分别在“中立”模式下按照不同的阈值切分为 CSV 格式和 Protobuf 格式，并分别对其全部读取，不同格式切分后的文件体积如表 5 和表 6 所示，测试读取的时间如图 10 a) 所示。可以看出，随着切分阈值的增加，两种格式的星表读取时间都逐渐减小直至达到稳定。Protobuf 格式在文件体积略大于 CSV 格式的情况下，当阈值大于 1 000 之后的星表读取速度比 CSV 格式快 50% 以上，基本达到硬盘连续读取 I/O 的速度。之后，本文又对这两种格式的星表进行了自交叉证认 (self-matching) 的测试，即先将未切分星表全部读入到内存并按 HEALPix 编号排序作为源星表，按天体在源星表中的顺序计算对应切分后目标星表文件的编号。若文件已存在于内存中，则与内存中的 KD-Tree 交叉证认；若未存在，则删除内存中的 KD-Tree，并读取对应的文件，建立新的 KD-Tree 进行交叉证认。统计不同阈值下的证认时间，结果如图 10 b) 所示。可以看出，随着阈值的

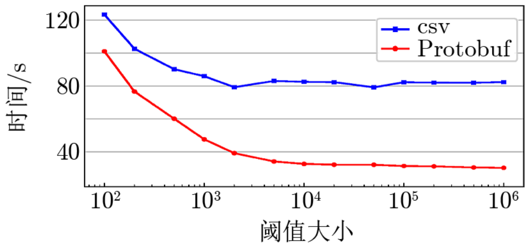
表 5　CSV 格式在不同阈值下切分后文件和原始星表的体积对比

阈值	单个文件平均体积/B	文件总数	总体积/B
100	59 804	59 804	3 406 675 650
200	112 524	30 275	3 406 675 650
500	287 192	11 862	3 406 675 650
1 000	546 555	6 233	3 406 675 650
2 000	1 136 316	2 998	3 406 675 650
5 000	2 695 155	1 264	3 406 675 650
10 000	5 584 714	610	3 406 675 650
20 000	10 679 234	319	3 406 675 650
50 000	25 422 953	134	3 406 675 650
100 000	50 098 171	68	3 406 675 650
200 000	89 649 359	38	3 406 675 650
500 000	200 392 685	17	3 406 675 650
1 000 000	262 051 973	13	3 406 675 650
原始星表	3 396 245 328	1	3 396 245 328

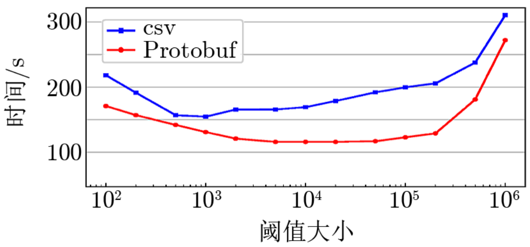
注：原始星表换行符是 \n，程序切分后星表换行符是 \r\n，导致文件总体积略有增加。

表 6　Protobuf 格式在不同阈值下切分后文件的体积对比

阈值	单个文件平均体积/B	文件总数	总体积/B
100	61 672	59 804	3 688 260 196
200	121 753	30 275	3 686 069 170
500	310 631	11 862	3 684 699 641
1 000	591 093	6 233	3 684 282 465
2 000	1 228 834	2 998	3 684 043 860
5 000	2 914 491	1 264	3 683 916 770
10 000	6 039 130	610	3 683 869 172
20 000	11 548 113	319	3 683 847 953
50 000	27 491 301	134	3 683 834 379
100 000	54 173 964	68	3 683 829 559
200 000	96 942 826	38	3 683 827 378
500 000	216 695 639	17	3 683 825 864
1 000 000	283 371 198	13	3 683 825 578



a)



b)

注：a) 不同阈值下星表读取速度曲线；b) 不同阈值下 Self-Matching 速度曲线。

图 10　对切分后 Protobuf 格式与 CSV 格式的星表读取和证认速度对比

增大，两种格式的星表证认时间都先减小后增大。CSV 格式的星表在阈值为 1 000 左右时速度最快，需要 155 s 左右；而 Protobuf 格式的星表在阈值为 10 000 时速度最快，只需约 116 s 即可完成。因此 Protobuf 格式对星表存储和交叉证认时的效率优于 CSV。

6.3 不同曲线与缓存算法在星表证认时的效率对比

对不同切分层级，分别用 HEALPix NESTED 模式下自带的 Z 曲线和 Peano-Hilbert 曲线遍历整个天区，在遍历到第 i 个区域时，同时读取 i 和以 i 为中心的上、下、左、右、左上、左下、右上、右下共九个区域的信息。若内存中已经缓存了该区域的信息，则为缓存命中；若不存在，则为缓存失效。此时将该区域的信息从磁盘中读取并加入到缓存，若缓存区已满，则调用替换算法选择一个区域替换，分别使用最久未使用 (LRU) 替换策略与先入先出 (FIFO) 缓存替换策略进行测试。

记录在不同层级和不同缓存大小的缓存失效次数，两种曲线和两种缓存算法的四种组合在四个不同的层级下的结果如图 11 所示。可以看出：

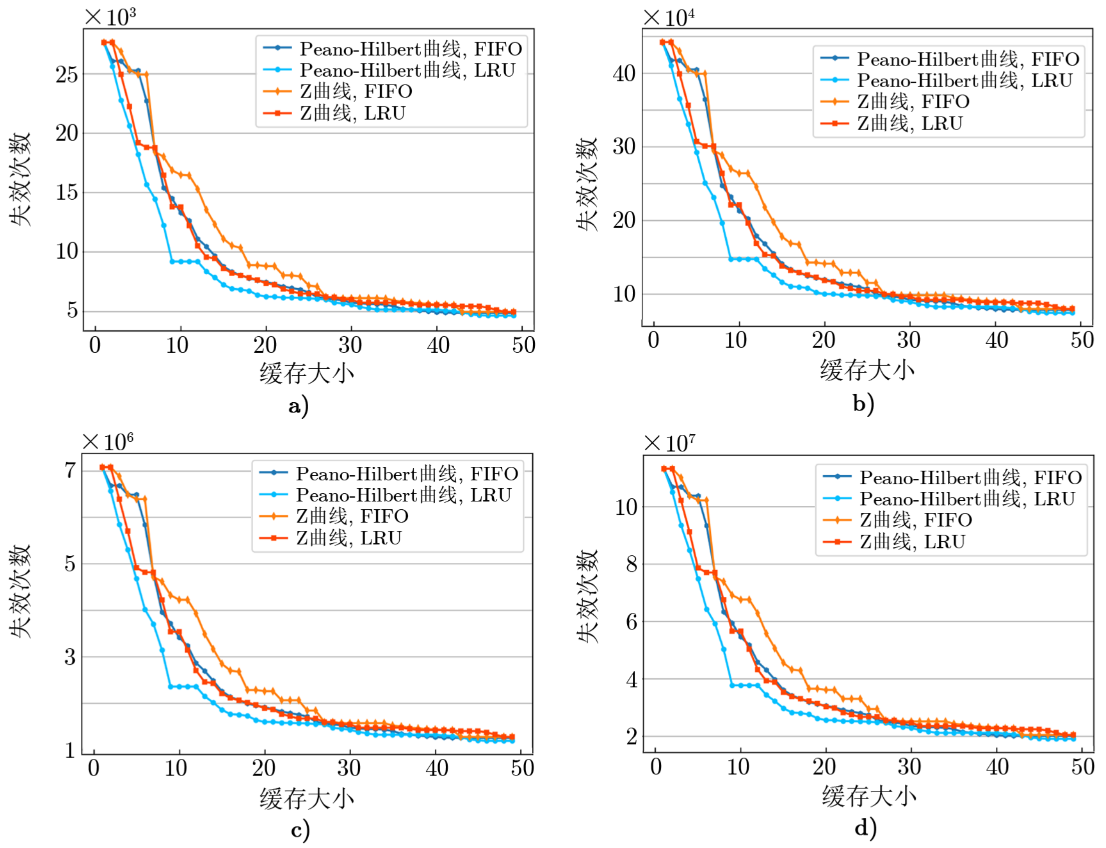


图 11　不同曲线和缓存替换算法在 $NSIDE$ =16, 64, 256, 1 024 时的失效次数曲线

1) 缓存越大，缓存失效的次数会越小，但单位缓存增加带来的性能提升也随之减少。

2) LRU 的效率要好于 FIFO，Peano-Hilbert 曲线的效率要好于 Z 曲线。

3) 遍历曲线和缓存替换算法的效率差距几乎不受 k 值大小的影响。

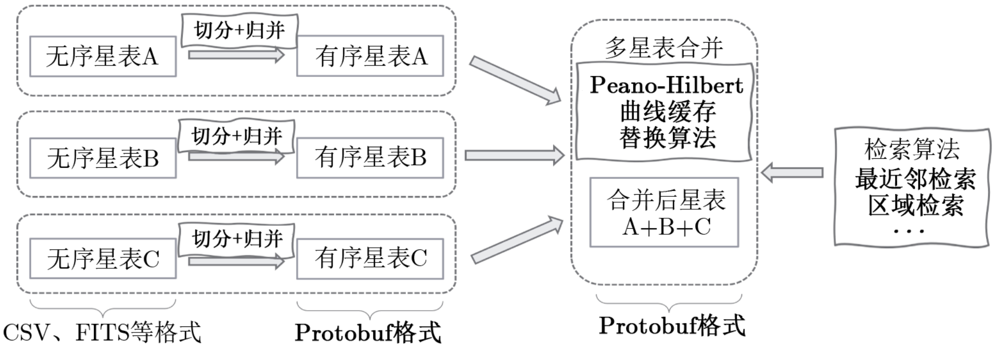
4) 少量的缓存大小就可以大幅度减少缓存失效的次数，提高效率。例如使用 Peano-Hilbert+LRU 方法，与不缓存 +FIFO 相比，当缓存大小为 9 时，就可以减少大约 66.7% 的读取次数；缓存大小增大到 20 时，则可以减少 77.4% 的读取次数。

5) 联合使用 Peano-Hilbert 曲线和 LRU 算法与 Z 曲线和 FIFO 之间的差异在缓存大小为 9 时达到最大，接近 1 倍。

由此可见，通过使用 Peano-Hilbert 曲线和 LRU 的缓存替换策略可以有效提升在星表融合时对天体交叉证认的效率。

7　结　论

针对大规模星表的高效检索问题，本文以文件作为存储的基础，提出了包括星表的切分、索引、合并在内的一系列解决方案。如图 12 所示，一种自适应密度的切分算法，能够在有限内存的情况下将大规模星表切分成多个限定大小范围的子星表，方便后续的检索等操作。经过测试，本文提出的“中立”模式切分算法可以将切分后的星表大小限制在 $[T, 4\times T]$ 之间，比其他模式更均匀，为星表高效检索奠定了基础。



注：加粗文字是本文创新部分。

图 12　本文提出的解决方案流程图

为了更快地检索星表，本文引入开源序列化库 Protocol Buffers 作为星表在磁盘中的存储格式。该库能够将天体坐标以二进制格式存储，在切分与检索时避免了浮点数与字符串之间的相互转换造成的性能消耗，提升了星表的读取效率。

在合并星表时，需要读取相邻星表以解决漏源问题。本文通过改进 HEALPix 的遍历方式，以 Peano-hilbert 曲线代替原有的 Z 曲线，并引入缓存和替换算法，减少了星表合并时的文件读取次数，提高了合并的速度。

在本文工作的基础上，未来可对诸多部分进行进一步优化与创新。例如，本文所述检索方法与测试结果是在单线程下取得，未来可将该方法扩展至多线程，或利用分布式集群等方

式将检索任务发送到多个子节点处理。另外，本文交叉证认的部分仅依靠坐标间角距离来判断是否同源，未来可将天体的亮度或其他参数作为判据引入，以提升在密集星场等复杂场合交叉证认的准确度。本文针对文件而设计诸多创新技术，未来可以尝试移植到数据库等其他非文件的场合，并加以优化，以期待实现更好的性能。

致谢

本论文得到国家天文科学数据中心、中国科学院天文科学数据中心、中国虚拟天文台、国家天文台-阿里云天文大数据联合研究中心提供的数据资源和技术支持。国家天文科学数据中心是由科技部、财政部认定的国家科技资源共享服务平台，属于基础支撑与条件保障类国家科技创新基地，依托单位是中国科学院国家天文台。

参考文献：

[1] Ivezić Ž, Kahn S M, Tyson J A, et al. ApJ, 2019, 873(2): 111
[2] Chambers K C, Magnier E A, Metcalfe N, et al. https://arxiv.org/abs/1612.05560, 2016
[3] 许允飞. 博士论文. 北京: 中国科学院国家天文台, 2020: 21
[4] Gray J, Nieto-Santisteban M A, Szalay A S. https://arxiv.org/abs/cs/0701171, 2007
[5] O'Mullane W, Banday A J, Górski K M, et al. Mining the Sky. Banday A J, Zaroubi S, Bartelmann M, eds. Heidelberg: Springer, 2001: 638
[6] Koposov S, Bartunov O. Astronomical Data Analysis Software and Systems XV. Gabriel C, Arviset C, Ponz D, et al, eds. San Francisco: ASP, 2006, 351: 735
[7] Górski K M, Hivon E, Banday A J, et al. ApJ, 2005, 622(2): 759
[8] Berriman G B, Good J C, Shiao B, et al. Astronomical Data Analysis Software and Systems XXVII. Ballester P, Ibsen J, Solar M, et al, eds. San Francisco: ASP, 2020, 522: 191
[9] Szalay A S, Gray J, Thakar A R, et al. https://arxiv.org/abs/cs/0202013, 2002
[10] Thakar A R, Szalay A, Fekete G, et al. Computing in Science & Engineering, 2008, 10(1): 30
[11] 高丹. 博士论文. 北京: 中国科学院国家天文台, 2008: 32
[12] Du P, Ren J, Pan J, et al. Science China Physics, Mechanics, and Astronomy, 2014, 57(3): 577
[13] Boch T, Pineau F, Derriere S. Astronomical Data Analysis Software and Systems XXI. Ballester P, Egret D, Lorente N P F, eds. San Francisco: ASP, 2012, 461: 291
[14] Pineau F X, Boch T, Derriere S. Astronomical Data Analysis Software and Systems XX. Evans I N, Accomazzi A, Mink D J, et al, eds. San Francisco: ASP, 2011, 442: 85
[15] Fernique P, Boch T, Donaldson T, et al. MOC-HEALPix Multi-Order Coverage map Version 1.0. https://www.ivoa.net/documents/MOC/20140602/index.html, 2014
[16] Fernique P, Allen M G, Boch T, et al. A&A, 2015, 578: A114
[17] Youngren R W, Petty M D. Heliyon, 2017, 3(6): 332
[18] Martinez-Castellanos I, Singer L P, Burns E, et al. AJ, 2022, 163(6): 259
[19] Hanisch R J, Farris A, Greisen E W, et al. A&A, 2001, 376: 359
[20] Wells D C, Greisen E W, Harten R H. A&AS, 1981, 44: 363
[21] de Berg M, van Kreveld M, Overmars M, et al. Computational Geometry: Algorithms and Applications. Heidelberg: Springer, 1997: 289
[22] 樊东卫, 何勃亮, 李长华, 等. 天文研究与技术, 2019, 16(1): 69
[23] Kleppmann M. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. Sebastopol: O'Reilly Media Inc., 2017: 112
[24] Varda K. Protocol buffers: Google's data interchange format. https://opensource.googleblog.com/2008/

07/protocol-buffers-googles-data.html, 2008

[25] Reinecke M, Hivon E. A&A, 2015, 580: A132

[26] Bern M, Eppstein D, Teng S H. International Journal of Computational Geometry & Applications, 1999, 9(06): 517

[27] Warren M S, Salmon J K. Proceedings of the 1993 ACM/IEEE conference on Supercomputing. New York: Association for Computing Machinery, 1993: 12

[28] Schäfer B M. Dissertation. München: LMU: Faculty of Physics, 2005: 99

The Efficient Indexing and Fusion Algorithms for Large-scale Catalogs Based on File

ZHANG Qi-qian[1,2], FAN Dong-wei[1,2,3], CUI Chen-zhou[1,2,3]

(1. National Astronomical Observatories, Chinese Academy of Sciences, Beijing 100101, China; 2. University of Chinese Academy of Sciences, Beijing 100049, China; 3. National Astronomical Data Center, Beijing 100101, China)

Abstract: Transient source searching, which aims at discovering changing objects in the sky, requires wide-field telescopes to survey the sky continuously. After sources are extracted on images, they will be cross-matched with existing large-scale catalogs to detect which object is changing. This step must be very fast, or it will slow down the data processing and cannot retrieve real-time discovery. But current referenced catalogs, e.g. SDSS, Gaia, Pan-STARRS, contain billions of objects. It is very difficult to complete the cross-matching step in seconds using traditional methods. In this paper, we propose a solution for the fast retrieval of hundreds of GB or even TB of catalogs with limited memory. Catalogs will be indexed in forms of individual files instead of database. A multi-resolution dynamic splitting algorithm based on HEALPix is introduced. It divides the catalogs into appropriate and uniform files according to the density of objects in different sky regions. A searching scheme is also designed with this algorithm. In order to improve the file reading speed, we create a medium storage mechanism to save file. The mechanism is based on Protocol Buffers, an open source component. The Peano-Hilbert curve is also applied to replace the HEALPix's original Z-curve, to fast traverse the catalog. With test, it effectively improves the cache hit ratio and data fusion efficiency on large-scale catalogs. With these improvements, billions objects searching and cross-matching in a limited hardware becomes possible. Our solution will help the implementation of changing object real-time detection, and other rapid detecting projects.

Key words: catalog; HEALPix; cross-matching; astronomy software; virtual observatory